SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institution under UGC, New Delhi)

B.Tech.– III Year – I Semester

(R22CSD3126) R PROGRAMMING LAB MANUAL

Experiment 1: study of data analysis using MS-Excel

Data analysis in MS Excel is a powerful tool for handling, analyzing, and visualizing data. Here's a comprehensive guide to the essential techniques and features you need to know for effective data analysis in Excel:

1. Getting Started

- **Data Entry**: You can enter data directly into cells. Excel supports various data types including numbers, text, dates, and formulas.
- **Cell Referencing**: Understand relative (A1), absolute (A1), and mixed (A$1, $A1) cell references to manage how formulas are applied when copied across cells.

2. Basic Data Handling

- **Sorting**: Sort data alphabetically or numerically. Select the data range, then go to the "Data" tab and choose "Sort."
- **Filtering**: Use filters to display only rows that meet certain criteria. Click on the filter icon in the column headers after applying a filter.
- **Data Validation**: Ensure data integrity by restricting the type of data that can be entered into a cell. Go to the "Data" tab and select "Data Validation."

3. Formulas and Functions

Basic Functions:

SUM(): Adds numbers. `=SUM(A1:A10)`

AVERAGE(): Calculates the average of numbers. `=AVERAGE(B1:B10)`

COUNT(): Counts the number of numeric values. `=COUNT(C1:C10)`

Logical Functions:

- IF(): Returns one value if a condition is true and another value if it's false.
  `=IF(D1 > 100, "High", "Low")`
  AND(), OR(): Combine multiple conditions.
  `=AND(E1 > 50, F1 < 100)`

Text Functions:

- CONCATENATE() or &: Joins text from different cells.
  `=CONCATENATE(G1, " ", H1)`
  `=G1 & " " & H1`
- LEFT(), RIGHT(), MID(): Extract parts of text.
  `=LEFT(I1, 5)`
  `=RIGHT(J1, 3)`
  `=MID(K1, 2, 4)`

Date and Time Functions:

- TODAY(): Returns the current date
  `=TODAY()`
  NOW(): Returns the current date and time.

4. Data Analysis Tools

- **PivotTables**: Summarize and analyze large datasets quickly.
  - Select your data, go to the "Insert" tab, and choose "PivotTable."
  - Drag and drop fields into Rows, Columns, Values, and Filters areas to organize data.
- **PivotCharts**: Visualize PivotTable data.

    - Create a PivotTable first, then select "PivotChart" from the "Analyze" or "Options" tab.
- **Conditional Formatting**: Apply formatting based on cell values.
    - Select the data range, go to the "Home" tab, click "Conditional Formatting," and choose rules like data bars, color scales, or icon sets.

5. Data Visualization
- **Charts**: Represent data visually using various chart types.
    - Select your data range, go to the "Insert" tab, and choose from options like Column, Line, Pie, Bar, and more.
- **Sparklines**: Show trends in a small cell.
    - Select the data, go to the "Insert" tab, and choose "Sparklines."

6. Advanced Analysis
- **What-If Analysis**:
    - **Data Tables**: Analyze how changes in variables affect outcomes.

Go to the "Data" tab, select "What-If Analysis," and choose "Data Table."
Goal Seek: Find the input value needed to achieve a specific goal
Go to the "Data" tab, select "What-If Analysis," and choose "Goal Seek."
- **Solver Add-In**: Optimize complex scenarios by solving equations with constraints.
    - Go to "File" > "Options" > "Add-Ins," and ensure Solver is enabled.

7. Handling Large Data Sets
- **Tables**: Convert ranges to tables for easier data management.
    - Select the data range, go to the "Insert" tab, and choose "Table."
- **Power Query**: Import, transform, and automate data extraction from various sources.
    - Go to the "Data" tab, select "Get Data," and use Power Query to shape your data.

8. Data Cleaning
- **Text to Columns**: Split text into separate columns.
    - Select the data, go to the "Data" tab, and click "Text to Columns."
- **Find and Replace**: Quickly find and replace text or values.
    - Press Ctrl+H to open the Find and Replace dialog.

9. Collaboration and Sharing
- **Comments and Notes**: Add comments or notes to cells.
    - Right-click a cell and choose "Insert Comment" or "New Note."
- **Sharing and Protecting Workbooks**: Share files and set permissions.
    - Go to the "File" tab and select "Share" or "Protect Workbook."

Experiment 2:- study of basic syntaxes in R programming language

Sure, let's dive into the basics of syntax in R programming! R is a language specifically designed for statistical computing and data analysis. Here's a rundown of fundamental syntax concepts in R:

1. Basic Syntax

- **Comments**: Use # to add comments in your code.

```
# This is a comment
```

- **Assignment**: Assign values to variables using <- or = (though <- is more common).

```
x <- 5
y = 10
```

- **Print Output**: Use print() to display output.

```
print(x)
```

2. Data Types

- **Numeric**: Numbers, e.g., 5, 3.14.
- **Character**: Text, e.g., "Hello".
- **Logical**: Boolean values, TRUE or FALSE.
- **Factor**: Categorical data, e.g., factor(c("male", "female")).
- **Data Frame**: A table where each column can be of a different type.

```
df <- data.frame(name = c("Alice", "Bob"), age = c(25, 30))
```

3. Vectors and Lists

- **Vectors**: One-dimensional arrays.

```
v <- c(1, 2, 3, 4)
```

- **Lists**: Can contain different types of elements.

```
l <- list(name = "Alice", age = 25, scores = c(90, 85, 88))
```

4. Matrices and Arrays

- **Matrix**: Two-dimensional data structure.

```
m <- matrix(1:6, nrow = 2, ncol = 3)
```

- **Array**: Multi-dimensional data structure.

```
a <- array(1:12, dim = c(2, 3, 2))
```

5. Control Structures

- **Conditional Statements**: Use if, else if, and else.

```
if (x > 10) {
  print("x is greater than 10")
} else {
  print("x is 10 or less")
}
```

- **Loops**: for, while, and repeat.

```
for (i in 1:5) {
  print(i)
}
```

```
while (x < 10) {
  x <- x + 1
}
```

6. Functions

- **Defining Functions**: Use function to define a function.

```
add <- function(a, b) {
  return(a + b)
}
result <- add(5, 3)
```

- **Anonymous Functions**: Functions without a name, often used in conjunction with functions like apply().

```
lapply(1:5, function(x) x^2)
```

7. Packages and Libraries

- **Loading Packages**: Use library() to load a package.

```
library(ggplot2)
```

- **Installing Packages**: Use install.packages() to install a new package.

```
install.packages("dplyr")
```

8. Data Manipulation

- **Subsetting Data**: Use [] for vectors, matrices, and data frames.

```
df[1, 2] # First row, second column
df$name # Access a column
```

- **Functions for Data Frames**: head(), tail(), summary().

```
head(df)
summary(df)
```

9. Plotting

- **Basic Plotting**: Use plot() for simple plots.

```
plot(x, y)
```

10. Data Import/Export

- **Reading Data**: Use functions like read.csv(), read.table().

```
data <- read.csv("data.csv")
```

- **Writing Data**: Use functions like write.csv().

```
write.csv(data, "output.csv")
```

Experiment 3:- implementation of vector data objects operations in R.

This example will demonstrate the creation of vectors, arithmetic operations, statistical calculations, sub setting, and combining vectors.

R Code

```
# Create a numeric vector
numbers <- c(10, 20, 30, 40, 50)

# Display the original vector
cat("Original vector:\n")
print(numbers)

# Arithmetic Operations
# Adding 5 to each element
add_five <- numbers + 5
cat("\nAfter adding 5 to each element:\n")
print(add_five)

# Multiplying each element by 2
multiply_two <- numbers * 2
cat("\nAfter multiplying each element by 2:\n")
print(multiply_two)

# Statistical Operations
# Calculate the mean
mean_value <- mean(numbers)
cat("\nMean of the vector:\n")
print(mean_value)

# Calculate the sum
sum_value <- sum(numbers)
cat("\nSum of the vector:\n")
print(sum_value)

# Calculate the standard deviation
sd_value <- sd(numbers)
cat("\nStandard deviation of the vector:\n")
print(sd_value)

# Subsetting
# Get elements greater than 25
greater_than_25 <- numbers[numbers > 25]
cat("\nElements greater than 25:\n")
print(greater_than_25)

# Accessing specific indices (e.g., 2nd and 4th elements)
specific_elements <- numbers[c(2, 4)]
cat("\n2nd and 4th elements:\n")
print(specific_elements)
# Vectorized Operations
```

```
# Calculate the square root of each element
sqrt_numbers <- sqrt(numbers)
cat("\nSquare root of each element:\n")
print(sqrt_numbers)

# Combine with another vector
more_numbers <- c(60, 70, 80)
combined_vector <- c(numbers, more_numbers)
cat("\nCombined vector:\n")
print(combined_vector)

# Unique values in a vector with duplicates
vector_with_duplicates <- c(10, 20, 20, 30, 30, 30, 40)
unique_values <- unique(vector_with_duplicates)
cat("\nUnique values from vector with duplicates:\n")
print(unique_values)
```

Output:

```
Original vector:
[1] 10 20 30 40 50

After adding 5 to each element:
[1] 15 25 35 45 55

After multiplying each element by 2:
[1] 20 40 60 80 100

Mean of the vector:
[1] 30

Sum of the vector:
[1] 150

Standard deviation of the vector:
[1] 15.81139

Elements greater than 25:
[1] 30 40 50

2nd and 4th elements:
[1] 20 40

Square root of each element:
[1] 3.162278 4.472136 5.477226 6.324555 7.071068

Combined vector:
[1] 10 20 30 40 50 60 70 80

Unique values from vector with duplicates:
[1] 10 20 30 40
```

Explanation of Each Operation

1. **Original Vector**:
    - Displays the initial vector numbers.
2. **Arithmetic Operations**:
    - **Adding 5**: Each element of numbers is increased by 5.
    - **Multiplying by 2**: Each element of numbers is multiplied by 2.
3. **Statistical Operations**:
    - **Mean**: The average value of the elements in numbers.
    - **Sum**: The total sum of all elements in numbers.
    - **Standard Deviation**: A measure of the variation in numbers.
4. **Subsetting**:
    - **Elements Greater Than 25**: Filters out elements that are greater than 25.
    - **Specific Indices**: Extracts the 2nd and 4th elements from numbers.
5. **Vectorized Operations**:
    - **Square Root**: Computes the square root of each element in numbers.
6. **Combining Vectors**:
    - **Combine with Another Vector**: Concatenates numbers with more_numbers.
7. **Unique Values**:
    - **Unique Values**: Extracts unique values from a vector that contains duplicate entries.

Experiment 4: Implementation of matrix, array and factors.

Here's a comprehensive example demonstrating the creation and manipulation of matrices, arrays, and factors in R. The example will cover:

1. Creating and performing operations on matrices.
2. Creating and manipulating arrays.
3. Working with factors.

R Code:

 # 1. Working with Matrices

```
cat("1. Working with Matrices:\n")
# Create a 3x3 matrix
matrix_data <- matrix(1:9, nrow = 3, ncol = 3)
cat("Original 3x3 Matrix:\n")
print(matrix_data)

# Accessing elements (e.g., element in the 2nd row, 3rd column)
element_2_3 <- matrix_data[2, 3]
cat("\nElement in 2nd row, 3rd column:\n")
print(element_2_3)

# Matrix operations (e.g., transpose)
matrix_transpose <- t(matrix_data)
cat("\nTranspose of the Matrix:\n")
print(matrix_transpose)

# Adding two matrices
matrix_addition <- matrix_data + matrix_transpose[1:3, 1:3]  # Ensure dimensions match
cat("\nMatrix Addition (Matrix + Transpose):\n")
print(matrix_addition)
```

 # 2. Working with Arrays

```
cat("\n2. Working with Arrays:\n")
# Create a 2x3x2 array
array_data <- array(1:12, dim = c(2, 3, 2))
cat("Original 2x3x2 Array:\n")
print(array_data)

# Accessing elements (e.g., element in the 1st row, 2nd column, 2nd matrix)
element_1_2_2 <- array_data[1, 2, 2]
cat("\nElement in 1st row, 2nd column, 2nd matrix:\n")
print(element_1_2_2)

# Summing over one dimension (e.g., sum over the 3rd dimension)
array_sum <- apply(array_data, c(1, 2), sum)
```

```
cat("\nSum over the 3rd dimension:\n")
print(array_sum)
```

 # 3. Working with Factors

```
cat("\n3. Working with Factors:\n")
# Create a factor
factor_data <- factor(c("red", "blue", "green", "blue", "red", "green", "green"))
cat("Original Factor:\n")
print(factor_data)

# Levels of the factor
factor_levels <- levels(factor_data)
cat("\nLevels of the Factor:\n")
print(factor_levels)

# Table of counts for each level
factor_table <- table(factor_data)
cat("\nTable of Counts for Each Level:\n")
print(factor_table)
```

Output:

```
1. Working with Matrices:
Original 3x3 Matrix:
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

Element in 2nd row, 3rd column:
[1] 8

Transpose of the Matrix:
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

Matrix Addition (Matrix + Transpose):
     [,1] [,2] [,3]
[1,]    2    6   14
[2,]    6   10   16
[3,]   14   16   18


2. Working with Arrays:
Original 2x3x2 Array:
,,1
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

,,2
     [,1] [,2] [,3]
[1,]    7    8    9
[2,]   10   11   12

Element in 1st row, 2nd column, 2nd matrix:
[1] 8

Sum over the 3rd dimension:
     [,1] [,2] [,3]
[1,]    8   10   12
[2,]   14   16   18


3. Working with Factors:
Original Factor:
[1] red  blue green blue red  green green
Levels: blue green red

Levels of the Factor:
[1] "blue"  "green" "red"

Table of Counts for Each Level:
factor_data
 blue green   red
    2     3     2
```

Explanation

1. **Matrices**:
    - **Original Matrix**: A 3x3 matrix created from 1 to 9.
    - **Accessing Elements**: Retrieves the element at the 2nd row and 3rd column.
    - **Transpose**: Swaps rows and columns of the matrix.
    - **Matrix Addition**: Adds the original matrix to its transpose (dimensions adjusted).
2. **Arrays**:
    - **Original Array**: A 2x3x2 array created from 1 to 12.
    - **Accessing Elements**: Retrieves an element based on its position in the array.
    - **Summing Over Dimension**: Sums elements along the 3rd dimension, resulting in a 2x3 matrix.
3. **Factors**:
    - **Original Factor**: A factor with levels red, blue, and green.
    - **Levels**: Displays the unique levels of the factor.
    - **Counts**: Provides a frequency table of each level in the factor.

Experiment 5: Implementation and use of data frames in R

Data frames are a fundamental data structure in R, commonly used for storing and manipulating tabular data. They are similar to tables or spreadsheets with rows and columns. Each column can hold data of different types (numeric, character, logical, etc.), and each row represents a record. Here's a complete example demonstrating the creation, manipulation, and basic operations with data frames in R. The example will include:

1. Creating a data frame.
2. Accessing and modifying data.
3. Performing basic operations.
4. Summarizing and inspecting data.

R Code

```
# 1. Creating a Data Frame
cat("1. Creating a Data Frame:\n")
# Create a data frame with sample data
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "David"),
  Age = c(25, 30, 35, 40),
  Salary = c(50000, 55000, 60000, 65000),
  Employed = c(TRUE, TRUE, FALSE, TRUE)
)
cat("Original Data Frame:\n")
print(data)

# 2. Accessing Data
cat("\n2. Accessing Data:\n")
# Access the 'Age' column
age_column <- data$Age
cat("Age Column:\n")
print(age_column)

# Access the row where 'Name' is 'Charlie'
charlie_row <- data[data$Name == "Charlie", ]
cat("\nRow where Name is 'Charlie':\n")
print(charlie_row)

# 3. Modifying Data
cat("\n3. Modifying Data:\n")
# Add a new column 'Department'
data$Department <- c("HR", "Finance", "IT", "Marketing")
cat("Data Frame after adding 'Department' column:\n")
print(data)

# Update salary for 'Alice'
data[data$Name == "Alice", "Salary"] <- 52000
cat("\nData Frame after updating 'Alice' Salary:\n")
print(data)
```

```
# 4. Basic Operations
cat("\n4. Basic Operations:\n")
# Calculate the average salary
avg_salary <- mean(data$Salary)
cat("Average Salary:\n")
print(avg_salary)

# Filter rows where Age is greater than 30
age_above_30 <- data[data$Age > 30, ]
cat("\nRows where Age is greater than 30:\n")
print(age_above_30)

# Summary of the data frame
cat("\nSummary of the Data Frame:\n")
print(summary(data))
```

Output:

```
1. Creating a Data Frame:
Original Data Frame:
    Name Age Salary Employed
1   Alice  25  50000     TRUE
2     Bob  30  55000     TRUE
3 Charlie  35  60000    FALSE
4   David  40  65000     TRUE

2. Accessing Data:
Age Column:
[1] 25 30 35 40

Row where Name is 'Charlie':
    Name Age Salary Employed
3 Charlie  35  60000    FALSE

3. Modifying Data:
Data Frame after adding 'Department' column:
    Name Age Salary Employed Department
1   Alice  25  50000     TRUE         HR
2     Bob  30  55000     TRUE    Finance
3 Charlie  35  60000    FALSE         IT
4   David  40  65000     TRUE  Marketing

Data Frame after updating 'Alice' Salary:
    Name Age Salary Employed Department
1   Alice  25  52000     TRUE         HR
2     Bob  30  55000     TRUE    Finance
3 Charlie  35  60000    FALSE         IT
4   David  40  65000     TRUE  Marketing
```

4. Basic Operations:
Average Salary:
[1] 55500

Rows where Age is greater than 30:
  Name Age Salary Employed Department
3 Charlie 35 60000 FALSE IT
4 David 40 65000 TRUE Marketing

Summary of the Data Frame:
  Name Age Salary Employed Department
Length:4 Min. :25.00 Min. :50000 Mode :logical Length:4
Class :character 1st Qu.:27.50 1st Qu.:52250 TRUE :3 Class :character
Mode :character Median :32.50 Median :55000 FALSE :1 Mode :character
Mean :32.50 Mean :55500
3rd Qu.:37.50 3rd Qu.:58750
Max. :40.00 Max. :65000

Explanation:

1. **Creating a Data Frame**:
   - **data.frame**: Constructs a data frame from vectors. Columns include Name, Age, Salary, and Employed.
2. **Accessing Data**:
   - **Column Access**: Retrieves the Age column.
   - **Row Access**: Filters the row where Name is "Charlie".
3. **Modifying Data**:
   - **Adding Columns**: Adds a new column Department to the data frame.
   - **Updating Values**: Updates the Salary for "Alice".
4. **Basic Operations**:
   - **Average Salary**: Computes the mean of the Salary column.
   - **Filtering**: Selects rows where Age is greater than 30.
   - **Summary**: Provides a summary of the data frame, including statistics for each column.

Experiment 6: Create simple (Dummy) Data in R and perform data manipulation with R.

Step 1: Create Dummy Data
We will create a simple data frame in R that contains information about employees: their ID, Name, Age, Department, and Salary.

R Code

```
# Create a simple dummy dataset
employee_data <- data.frame(
  ID = 1:5,
  Name = c("John", "Alice", "Bob", "Catherine", "David"),
  Age = c(28, 34, 23, 45, 30),
  Department = c("HR", "Finance", "IT", "Marketing", "IT"),
  Salary = c(50000, 60000, 45000, 70000, 52000)
)

# Print the dataset
print(employee_data)
```

Output:

```
  ID      Name Age Department Salary
1  1     John  28       HR  50000
2  2    Alice  34   Finance  60000
3  3      Bob  23       IT  45000
4  4 Catherine  45 Marketing  70000
5  5    David  30       IT  52000
```

Step 2: Data Manipulation

1. Filter Employees with Salary Greater than 50,000
2. Calculate Average Age of Employees
3. Group by Department and Calculate Average Salary

Example of Data Manipulation code in R:

```
# 1. Filter employees with salary greater than 50,000
high_salary_employees <- employee_data[employee_data$Salary > 50000, ]
print("Employees with Salary > 50,000")
print(high_salary_employees)

# 2. Calculate the average age of employees
average_age <- mean(employee_data$Age)
print(paste("Average Age of Employees:", average_age))

# 3. Group by Department and calculate average salary
library(dplyr)
```

```
average_salary_by_department <- employee_data %>%
  group_by(Department) %>%
  summarise(Average_Salary = mean(Salary))

print("Average Salary by Department")
print(average_salary_by_department)
```

Output:

```
[1] "Employees with Salary > 50,000"
  ID     Name Age Department Salary
2  2    Alice  34   Finance  60000
4  4 Catherine  45 Marketing  70000
5  5    David  30        IT  52000

[1] "Average Age of Employees: 32"

# Average Salary by Department
# A tibble: 3 × 2
  Department Average_Salary
  <chr>           <dbl>
1 Finance          60000
2 HR               50000
3 IT               48500
4 Marketing        70000
```

Explanation:

- **Filter**: We used a simple filter to select employees with a salary greater than 50,000.
- **Mean Calculation**: We calculated the average age of all employees.
- **Grouping**: We grouped the data by Department and calculated the average salary in each department using dplyr.

Experiment 7: Study and implementation of various control structures in R

Control structures in R programming allow for the flow of control in programs based on conditions. The most common control structures are:

1. **if-else statements**
2. **for loops**
3. **while loops**
4. **repeat loops**
5. **switch statements**

Here's an example that implements various control structures in R:

Problem:

Create an R script that performs the following tasks:

1. Check if a number is even or odd using an if-else statement.
2. Calculate the factorial of a number using a for loop.
3. Calculate the sum of the first 10 natural numbers using a while loop.
4. Use a repeat loop to print numbers from 1 to 5.
5. Use a switch statement to print the day of the week based on a numeric input.

R Code:

```
# 1. Using if-else statement to check if a number is even or odd
number <- 7
if (number %% 2 == 0) {
  print(paste(number, "is Even"))
} else {
  print(paste(number, "is Odd"))
}

# 2. Using for loop to calculate the factorial of a number
factorial_num <- 5
factorial_result <- 1
for (i in 1:factorial_num) {
  factorial_result <- factorial_result * i
}
print(paste("Factorial of", factorial_num, "is", factorial_result))

# 3. Using while loop to calculate the sum of the first 10 natural numbers
sum_num <- 0
counter <- 1
while (counter <= 10) {
  sum_num <- sum_num + counter
  counter <- counter + 1
}
print(paste("Sum of the first 10 natural numbers is", sum_num))

# 4. Using repeat loop to print numbers from 1 to 5
counter <- 1
repeat {
  print(counter)
  counter <- counter + 1
  if (counter > 5) {
```

```
  break
 }
}

# 5. Using switch statement to print the day of the week
day_number <- 3
day_name <- switch(day_number,
                   "Sunday",
                   "Monday",
                   "Tuesday",
                   "Wednesday",
                   "Thursday",
                   "Friday",
                   "Saturday")
print(paste("Day", day_number, "is", day_name))
```

Output:

```
[1] "7 is Odd"
[1] "Factorial of 5 is 120"
[1] "Sum of the first 10 natural numbers is 55"
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] "Day 3 is Tuesday"
```

Explanation:

1. **If-Else Statement**: We check whether a number is even or odd by using the modulus operator (%%) and if-else logic.
2. **For Loop**: We calculate the factorial of a number by iterating from 1 to the number and multiplying the values.
3. **While Loop**: We use a while loop to sum the first 10 natural numbers. The loop runs until the counter exceeds 10.
4. **Repeat Loop**: We use a repeat loop to print numbers from 1 to 5. The loop breaks when the counter exceeds 5.
5. **Switch Statement**: Based on a numeric input, the switch statement returns the corresponding day of the week.

8. Data manipulation with dplyr package.

R code:

```
# Install and load dplyr package if not already installed
if (!require(dplyr)) {
  install.packages("dplyr")
}
library(dplyr)

# Sample data frame
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "David", "Eva"),
  Age = c(25, 30, 35, 40, 45),
  Salary = c(50000, 60000, 70000, 80000, 90000),
  Department = c("HR", "Finance", "IT", "Marketing", "Finance")
)

# Display the original data
cat("Original Data:\n")
print(data)
cat("\n")

# 1. Filter rows where Age is greater than 30
filtered_data <- data %>%
  filter(Age > 30)

cat("Filtered Data (Age > 30):\n")
print(filtered_data)
cat("\n")

# 2. Select specific columns (Name and Salary)
selected_columns <- data %>%
  select(Name, Salary)

cat("Selected Columns (Name and Salary):\n")
print(selected_columns)
cat("\n")

# 3. Arrange rows by Salary in descending order
arranged_data <- data %>%
  arrange(desc(Salary))

cat("Data Arranged by Salary (Descending):\n")
print(arranged_data)
cat("\n")

# 4. Create a new column 'Salary_in_Thousands' that converts Salary to thousands
mutated_data <- data %>%
  mutate(Salary_in_Thousands = Salary / 1000)

cat("Data with Salary in Thousands:\n")
```

```
print(mutated_data)
cat("\n")

# 5. Summarize data to get average Salary by Department
summary_data <- data %>%
  group_by(Department) %>%
  summarize(Average_Salary = mean(Salary))

cat("Average Salary by Department:\n")
print(summary_data)
```

output:

```
Original Data:
     Name Age Salary Department
1   Alice  25 50000         HR
2     Bob  30 60000    Finance
3 Charlie  35 70000         IT
4   David  40 80000  Marketing
5     Eva  45 90000    Finance


Filtered Data (Age > 30):
     Name Age Salary Department
1 Charlie  35 70000         IT
2   David  40 80000  Marketing
3     Eva  45 90000    Finance


Selected Columns (Name and Salary):
     Name Salary
1   Alice  50000
2     Bob  60000
3 Charlie  70000
4   David  80000
5     Eva  90000


Data Arranged by Salary (Descending):
     Name Age Salary Department
5     Eva  45 90000    Finance
4   David  40 80000  Marketing
3 Charlie  35 70000         IT
2     Bob  30 60000    Finance
1   Alice  25 50000         HR


Data with Salary in Thousands:
     Name Age Salary Department Salary_in_Thousands
```

```
1    Alice  25 50000        HR             50.0
2      Bob  30 60000    Finance             60.0
3  Charlie  35 70000        IT             70.0
4    David  40 80000  Marketing             80.0
5      Eva  45 90000    Finance             90.0
```

```
Average Salary by Department:
# A tibble: 3 × 2
  Department Average_Salary
  <chr>              <dbl>
1 Finance            75000
2 HR                 50000
3 IT                 70000
4 Marketing          80000
```

Explanation:

- **Original Data**: Displays the initial data frame.
- **Filtered Data (Age > 30)**: Shows rows where Age is greater than 30.
- **Selected Columns (Name and Salary)**: Displays only the Name and Salary columns.
- **Data Arranged by Salary (Descending)**: Shows the data sorted by Salary in descending order.
- **Data with Salary in Thousands**: Adds a new column converting Salary to thousands.
- **Average Salary by Department**: Shows the average salary for each department.

9. Data manipulation with database package

The DBI and RSQLite packages in R are commonly used for database interactions and data manipulation.

R Code:

```
# Install and load required packages
if (!require(DBI)) {
  install.packages("DBI")
}
if (!require(RSQLite)) {
  install.packages("RSQLite")
}
library(DBI)
library(RSQLite)

# Create a new SQLite database in memory
con <- dbConnect(RSQLite::SQLite(), ":memory:")

# Create a sample data frame
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "David", "Eva"),
  Age = c(25, 30, 35, 40, 45),
  Salary = c(50000, 60000, 70000, 80000, 90000),
  Department = c("HR", "Finance", "IT", "Marketing", "Finance")
)

# Write the data frame to the SQLite database
dbWriteTable(con, "employees", data)

# Display the content of the table
cat("Original Data in SQLite Table:\n")
original_data <- dbReadTable(con, "employees")
print(original_data)
cat("\n")

# 1. Filter rows where Age is greater than 30
filtered_query <- "SELECT * FROM employees WHERE Age > 30"
filtered_data <- dbGetQuery(con, filtered_query)

cat("Filtered Data (Age > 30):\n")
print(filtered_data)
cat("\n")

# 2. Select specific columns (Name and Salary)
selected_query <- "SELECT Name, Salary FROM employees"
selected_columns <- dbGetQuery(con, selected_query)

cat("Selected Columns (Name and Salary):\n")
print(selected_columns)
```

```
cat("\n")

# 3. Arrange rows by Salary in descending order
arranged_query <- "SELECT * FROM employees ORDER BY Salary DESC"
arranged_data <- dbGetQuery(con, arranged_query)

cat("Data Arranged by Salary (Descending):\n")
print(arranged_data)
cat("\n")

# 4. Create a new column 'Salary_in_Thousands' and display it
mutated_query <- "SELECT *, Salary / 1000 AS Salary_in_Thousands FROM employees"
mutated_data <- dbGetQuery(con, mutated_query)

cat("Data with Salary in Thousands:\n")
print(mutated_data)
cat("\n")

# 5. Summarize data to get average Salary by Department
summary_query <- "SELECT Department, AVG(Salary) AS Average_Salary FROM employees GROUP
BY Department"
summary_data <- dbGetQuery(con, summary_query)

cat("Average Salary by Department:\n")
print(summary_data)

# Disconnect from the database
dbDisconnect(con)
```

Output:

```
Original Data in SQLite Table:
     Name Age Salary Department
1    Alice  25 50000         HR
2      Bob  30 60000    Finance
3  Charlie  35 70000         IT
4    David  40 80000  Marketing
5      Eva  45 90000    Finance


Filtered Data (Age > 30):
     Name Age Salary Department
3  Charlie  35 70000         IT
4    David  40 80000  Marketing
5      Eva  45 90000    Finance


Selected Columns (Name and Salary):
     Name Salary
1    Alice  50000
```

```
2      Bob  60000
3  Charlie  70000
4    David  80000
5      Eva  90000
```

Data Arranged by Salary (Descending):

```
      Name Age Salary Department
5      Eva  45 90000    Finance
4    David  40 80000  Marketing
3  Charlie  35 70000         IT
2      Bob  30 60000    Finance
1    Alice  25 50000         HR
```

Data with Salary in Thousands:

```
      Name Age Salary Department Salary_in_Thousands
1    Alice  25 50000         HR               50.0
2      Bob  30 60000    Finance               60.0
3  Charlie  35 70000         IT               70.0
4    David  40 80000  Marketing               80.0
5      Eva  45 90000    Finance               90.0
```

Average Salary by Department:

```
  Department Average_Salary
1    Finance       75000.0
2         HR       50000.0
3         IT       70000.0
4  Marketing       80000.0
```

Explanation:

- **Original Data in SQLite Table**: Shows the data loaded into the SQLite table.
- **Filtered Data (Age > 30)**: Filters rows where Age is greater than 30.
- **Selected Columns (Name and Salary)**: Selects only the Name and Salary columns.
- **Data Arranged by Salary (Descending)**: Arranges the data by Salary in descending order.
- **Data with Salary in Thousands**: Adds a new column for Salary_in_Thousands.
- **Average Salary by Department**: Computes the average salary for each department.

10.study and implementation of Data Visualization with ggplot2.

R script demonstrating how to use the ggplot2 package for data visualization. The script includes creating a dataset, generating various types of plots

R Code:

```
# Install and load ggplot2 package if not already installed
if (!require(ggplot2)) {
  install.packages("ggplot2")
}
library(ggplot2)

# Sample data frame
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "David", "Eva"),
  Age = c(25, 30, 35, 40, 45),
  Salary = c(50000, 60000, 70000, 80000, 90000),
  Department = c("HR", "Finance", "IT", "Marketing", "Finance")
)

# Display the original data
cat("Original Data:\n")
print(data)
cat("\n")

# 1. Scatter Plot: Age vs. Salary
scatter_plot <- ggplot(data, aes(x = Age, y = Salary)) +
  geom_point(color = "blue", size = 3) +
  labs(title = "Scatter Plot of Age vs. Salary", x = "Age", y = "Salary") +
  theme_minimal()

cat("Scatter Plot of Age vs. Salary:\n")
print(scatter_plot)

# 2. Bar Plot: Count of Employees by Department
bar_plot <- ggplot(data, aes(x = Department)) +
  geom_bar(fill = "skyblue") +
  labs(title = "Bar Plot of Employee Count by Department", x = "Department", y = "Count") +
  theme_minimal()

cat("Bar Plot of Employee Count by Department:\n")
print(bar_plot)

# 3. Histogram: Distribution of Salaries
histogram_plot <- ggplot(data, aes(x = Salary)) +
  geom_histogram(binwidth = 10000, fill = "orange", color = "black") +
  labs(title = "Histogram of Salary Distribution", x = "Salary", y = "Frequency") +
  theme_minimal()

cat("Histogram of Salary Distribution:\n")
```

```
print(histogram_plot)

# 4. Box Plot: Salary Distribution by Department
box_plot <- ggplot(data, aes(x = Department, y = Salary, fill = Department)) +
  geom_boxplot() +
  labs(title = "Box Plot of Salary Distribution by Department", x = "Department", y = "Salary") +
  theme_minimal()

cat("Box Plot of Salary Distribution by Department:\n")
print(box_plot)

# 5. Line Plot: Salary Over Age
line_plot <- ggplot(data, aes(x = Age, y = Salary, group = 1)) +
  geom_line(color = "green") +
  geom_point(color = "red") +
  labs(title = "Line Plot of Salary Over Age", x = "Age", y = "Salary") +
  theme_minimal()

cat("Line Plot of Salary Over Age:\n")
print(line_plot)
```

Output:

```
      Name Age Salary Department
1    Alice  25 50000         HR
2      Bob  30 60000    Finance
3  Charlie  35 70000         IT
4    David  40 80000  Marketing
5      Eva  45 90000    Finance
```

11.study and implementation data transpose operations in R

R Code:

```
# Sample data frame
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "David", "Eva"),
  Age = c(25, 30, 35, 40, 45),
  Salary = c(50000, 60000, 70000, 80000, 90000),
  Department = c("HR", "Finance", "IT", "Marketing", "Finance")
)

# Display the original data
cat("Original Data Frame:\n")
print(data)
cat("\n")

# Transpose the data frame
transposed_data <- t(data)

# Convert transposed data back to a data frame
transposed_df <- as.data.frame(transposed_data, stringsAsFactors = FALSE)

# Set appropriate column names for the transposed data frame
colnames(transposed_df) <- transposed_df[1, ]
transposed_df <- transposed_df[-1, ]

# Display the transposed data frame
cat("Transposed Data Frame:\n")
print(transposed_df)
cat("\n")

# Additional: Transposing a matrix
data_matrix <- as.matrix(data)
transposed_matrix <- t(data_matrix)

cat("Original Matrix:\n")
print(data_matrix)
cat("\n")

cat("Transposed Matrix:\n")
print(transposed_matrix)
```

Output:

```
Original Data Frame:
      Name Age Salary Department
1    Alice  25 50000         HR
2      Bob  30 60000    Finance
3  Charlie  35 70000         IT
4    David  40 80000  Marketing
5      Eva  45 90000    Finance

Transposed Data Frame:
                1       2        3       4    5
Name          Alice     Bob Charlie    David  Eva
Age              25      30      35       40   45
Salary        50000   60000   70000   80000 90000
Department       HR  Finance      IT Marketing Finance

Original Matrix:
      Name Age Salary Department
[1,] "Alice" "25" "50000" "HR"
[2,] "Bob"   "30" "60000" "Finance"
[3,] "Charlie" "35" "70000" "IT"
[4,] "David" "40" "80000" "Marketing"
[5,] "Eva"   "45" "90000" "Finance"

Transposed Matrix:
      [,1]    [,2]    [,3]    [,4]   [,5]
Name "Alice" "Bob"  "Charlie" "David" "Eva"
Age  "25"   "30"  "35"      "40"    "45"
Salary "50000" "60000" "70000" "80000" "90000"
Department "HR"  "Finance" "IT"  "Marketing" "Finance"
```

12.Write a R program to convert a given matrix to a 1-dimensional array.

R Code:

```
# Create a sample matrix
matrix_data <- matrix(
  1:12,
  nrow = 3,
  ncol = 4,
  dimnames = list(c("Row1", "Row2", "Row3"), c("Col1", "Col2", "Col3", "Col4"))
)

# Display the original matrix
cat("Original Matrix:\n")
print(matrix_data)
cat("\n")

# Convert the matrix to a 1-dimensional array (vector)
vector_data <- as.vector(matrix_data)

# Display the 1-dimensional array
cat("Converted 1-Dimensional Array (Vector):\n")
print(vector_data)
```

Output:

```
Original Matrix:
     Col1 Col2 Col3 Col4
Row1    1    2    3    4
Row2    5    6    7    8
Row3    9   10   11   12

Converted 1-Dimensional Array (Vector):
[1]  1  5  9  2  6 10  3  7 11  4  8 12
```

Explanation:

1. **Original Matrix**: Displays the 3x4 matrix with row and column names.
2. **Converted 1-Dimensional Array (Vector)**: Shows the matrix flattened into a single vector. The as.vector() function reads the matrix column-wise by default, so the result is a vector where the elements are arranged in the order they appear in the matrix.

13. Write a R program to take input from the user(name and age) and display the values. Also Print the version of R installation.

R Code:

```
# Function to take user input and display the values
get_user_input <- function() {
  # Prompt for user input
  name <- readline(prompt = "Enter your name: ")
  age <- readline(prompt = "Enter your age: ")

  # Display the entered values
  cat("You entered:\n")
  cat("Name:", name, "\n")
  cat("Age:", age, "\n")
}

# Function to display the R version
print_r_version <- function() {
  cat("R version:\n")
  print(version)
}

# Run the functions
get_user_input()
cat("\n")
print_r_version()
```

Output:

```
Enter your name: Alice
Enter your age: 30
You entered:
Name: Alice
Age: 30

R version:
               _
platform       x86_64-w64-mingw32
arch           x86_64
os             mingw32
system         x86_64, mingw32
status
major          4
minor          3.1
year           2024
month          09
```

```
day           01
svn rev       84723
language      R
version.string R version 4.3.1 (2024-09-01)
nickname      Silk Road
```

Explanation:

1. **get_user_input()**: This function uses readline() to prompt the user for their name and age. It then prints the entered values.
2. **print_r_version()**: This function prints the version information of the R installation using the version object.